Accepted Manuscript

Semi-online task assignment policies for workload consolidation in cloud computing systems

Vincent Armant, Milan De Cauwer, Kenneth N. Brown, Barry O'Sullivan

PII: S0167-739X(17)31914-3
DOI: https://doi.org/10.1016/j.future.2017.12.035
Reference: FUTURE 3873

To appear in: *Future Generation Computer Systems*

Received date : 1 September 2017
Revised date : 11 December 2017
Accepted date : 22 December 2017

Please cite this article as: V. Armant, M.D. Cauwer, K.N. Brown, B. O'Sullivan, Semi-online task assignment policies for workload consolidation in cloud computing systems, *Future Generation Computer Systems* (2018), https://doi.org/10.1016/j.future.2017.12.035

Semi-Online Task Assignment Policies for Workload Consolidation in Cloud Computing Systems

Vincent Armant, Milan De Cauwer, Kenneth N. Brown, Barry O'Sullivan

Insight Centre for Data Analytics
Department of Computer Science, University College Cork, Ireland
{vincent.armant|milan.decauwer|ken.brown|barry.osullivan}@insight-centre.org

Abstract

Satisfying on-demand access to cloud computing infrastructures under quality-of-service constraints while minimising the wastage of resources is an important challenge in data centre resource management. In this paper we tackle this challenge in a semi-online workload management system allocating tasks with uncertain duration to physical servers. Our semi-online framework, based on a bin packing approach, allows us to gather information on incoming tasks during a short time window before deciding on their assignments. Our contributions are as follows: (i) we propose a formal framework capturing the semi-online consolidation problem; (ii) we propose a new dynamic and real-time allocation algorithm based on the incremental merging of bins; and (iii) an adaptation of standard bin packing heuristics with a local search algorithm for the semi-online context considered here. We provide a systematic study of the impact of varying time-period size and varying the degrees of uncertainty on the duration of incoming tasks. The policies are compared in terms of solution quality and solving time on a data-set extracted from a real-world cluster trace.

Our results show that, around periods of high demand, our best policy saves up to 40% of the resources compared to the other polices, and is robust to uncertainty in the task durations. Finally, we show that small increases in the allowable time window allows a significant improvement, but that larger time windows do not necessarily improve resource usage for real world data sets.

Keywords: Cloud Computing, Workload consolidation, Semi-Online policies, Stochastic task duration.

1. Introduction

As the demand for IT services continues to increase, worldwide deployment of large data centres is continuing to grow. Those data centres consume enormous amounts of electricity, estimated at 70 terawatt hours for the USA alone in 2014, at a cost of 7 billion dollars [1]. It has been estimated that only 6 to 12% of electricity used by data centres can be attributed to productive computation [2]. Opportunities for reducing the energy consumption of data centres include more efficient cooling, enhanced power management for idle running, and aggressive resource sharing through virtualization.

The latter strategy is the one explored in this paper. The aim of resource consolidation through virtualization is to increase the utilisation of a subset of servers. Consolidation is usually achieved by allocating multiple tasks on the same physical machine. In turn, workload consolidation allows data centre operators to spread workload over a smaller set of machines so that those remaining unused can be powered down or put into a standby mode. Data centres are usually over-provisioned so they can cope with high fluctuations in demand from clients. Having a much larger pool of servers than needed also allows the design of fault-resilient systems [2]. As a consequence of over-provisioning, the workload can be dispatched without delays due to resource scarcity. In such a context one aims to dispatch tasks on machines so that resource wastage is minimised.

We leverage semi-online optimisation techniques in which workload allocations must be made without full knowledge of future demands. A semi-online formulation of the workload consolidation problem gathers information on incoming tasks for a short period of time. It may allow an operator to take more informed decisions than the fully online formulation while keeping control of delays in task deployments. In a cloud computing production environment, it is often the case that the duration for which a task will lock resources is either approximated or not known at all. This fits the new challenge of on-demand allocations in which demands are guaranteed to be satisfied in real-time. We therefore formalise the workload consolidation problem as a semi-online bin-packing problem whereby each bin maps to a machine and each item maps to a task.

The remainder of this paper is organised as follows. Section 2 discusses relevant work from both the cloud computing and optimisation communities. In Section 3 we provide a mathematical formulation of the on-demand bin packing (ODBP) problem. Section 4 introduces a novel methodology based on bin merging to solve the ODBP. Section 5 shows the performance of our approach compare to adapted heuristics of the related work in terms of solving time and solution quality. We demonstrate that our bin merging policy can achieve reductions in energy use of up to 40% over the comparator approaches. We show that the policy is relatively robust to increased errors in the predicted duration of the tasks. Finally, we show that moving from the pure online problem to the semi-online problem, with relatively small decision time windows, has a significant impact on the solution quality, but that all policies quickly stabilise and do not benefit further from longer time windows.

2. Related Work

Workload consolidation in data centres (DC) is a key challenge in the operations of cloud computing systems through which operators efficiently dispatch a workload on a pool of servers over which operations are virtualised [3]. Workload consolidation aims at maximising the usage of servers by grouping tasks to run concurrently on fewer machines. This technique is used to maintain control over the potentially high economic and environmental cost [4]. Due to the large spectrum of technologies that implement cloud computing systems, there is a vast body of literature reviewing efforts to optimise task placement (see surveys [5, 6, 7]). Depending on the technologies at play, this can be achieved either dynamically or statically. In the dynamic version, the

DC management system is allowed to migrate tasks across hosts [8, 9]. On the other hand, static workload consolidation does not allow migrations and focuses on consolidating the initial task placement. The operational setting in which our work stands is *on-demand* static placement of tasks. Moreover an efficient placement of tasks may reduce the cost of migrating tasks from a physical host to an other.

Policies for static consolidation have been studied in [10]. The authors leverage machine learning methods to predict the often unknown size of the virtual machines (VMs). They show that different prediction methods lead to assignments with significant differences in terms of resource usage. Our study takes place after this prediction step. We suppose that the size of each task can be predicted by the consolidation system upon its arrival. In the context of cloud gaming, the authors of [11] have considered the play-request dispatching problem where the aim is to minimise the total rental cost of a cloud platform. Although this problem has strong connections to ours, it differs in that it assumes that each task has the same size as well as perfect information concerning the their duration.

We tackle the semi-online formulation of the on-demand workload consolidation where tasks have to be allocated to servers in real-time. While the vast majority of the work carried on workload consolidation considers either the offline [12, 13] or online [14, 15, 16] setting, we cast the problem in a semi-online framework by considering a short period of a few seconds within which tasks are grouped before being allocated to hosts. More precisely, we build upon the offline workload consolidation problem discussed in [17].

In classical bin packing (BP) (see survey [18]) we are given a set of items along with their sizes and a set of bins having equal capacities. The objective is to find an assignment from items to bins such that the total number of bins is minimised. Many variants of BP have been studied extensively in both offline and online settings. Van Hentenryck et al [19] considered a number of different approaches to online stochastic optimisation under time constraints, including various packing models. The authors exploited distributions over future task arrivals. Delaying or rejecting tasks is allowed but neither of which applies to the problem we address. The semi-online bin packing problem, also known as batch bin packing, has been described in [20] in which the authors derive lower bounds on the minimum number of bins to be used to accommodate tasks over time. That work does not model the relationship between batches induced by item durations.

Several variants of BP are concerned with items that have a 'lifespan' within the bins. In offline dynamic BP (DBP), we are given items defined by their size and lifespan. The objective is to minimise the maximum number of bins over a given horizon. Here, we study the problem from a different viewpoint where we aim to minimise the cumulative cost. As an extension, fully DBP [21, 22] considers rearranging the items across the bins to retain a minimal number of used bins. In our study, instead of rearranging tasks to optimise the consolidation, we explore the possibility of gathering the tasks during a short time window to make more informed decisions for their placement. In [11], the authors revisited the online version of the dynamic BP problem in which items are characterised by size, arrival time, and an arbitrary departure time with the objective being to minimise the maximum number of bins ever used over time.

Finally, the family of online scheduling problems are to some extent related to our

work. In [23], the authors propose and evaluate online scheduling policies and define new distance metrics used in the best-Fit family of heuristics. The objective function under consideration is to minimise the queuing delay under constrained overall computational capacity. In our study, we propose a new approach for the semi online context that enforces a fixed delay satisfying the on demand placement. We also compare the efficiency of this heuristic against adapted Bin Packing heuristics.

3. Minimising Resource Wastage

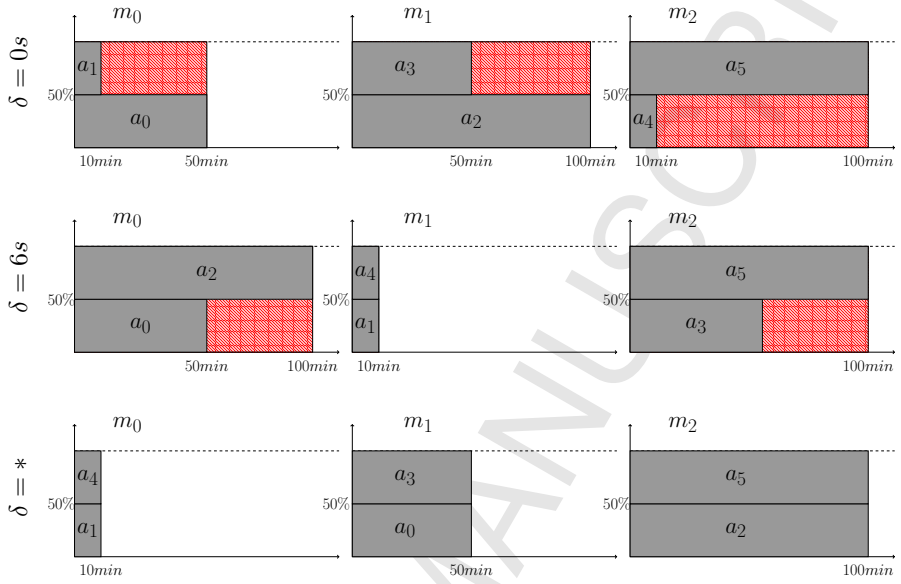
On-line policies decide the placement of incoming tasks as soon as they arrive in the system. Such a framework must fully satisfy on-demand Quality-of-Service (QoS) requirements, guaranteeing the real-time placements of tasks. However, due to the lack knowledge of which tasks might come next, on-line placement strategies may provide poor consolidation solutions and waste more resources than required.

On the other hand, for efficient resource utilisation, off-line approaches consider the task placement as a batch optimisation problem for which the incoming tasks are known in advance. The knowledge of forthcoming tasks allows a better consolidation but involves more sophisticated techniques that may required an expensive solving time. In the context of on-demand placement, neither the existence of incoming tasks nor their duration can be known in advance.

The aim of our approach is to fill the gap between on-line and off-line approaches and investigate the benefit of a semi-online framework in terms of the trade-off between efficient resource utilisation and on-demand placement QoS. To have a better understanding of our overall objective in the semi-online context, consider an arbitrary start time of 0. For any time t, let z_m^t be the observed run time duration for machine m between the time points 0 and t. Then, our overall objective is, for some sufficiently large time t, to allocate tasks to machines such that each task starts within δ seconds of its arrival time, and so that the sum over m of z_m^t is minimised.

Let us consider the following example of a sequence of six incoming tasks $a_0 ... a_5$, having respectively an expected duration of 50, 10, 100, 50, 10, 100 minutes, requiring the same CPU resource equivalent to 50% of a machine capacity, and arriving at one second of intervals. We aim to minimise the run-time of allocated machines. Figure 1 shows an optimal placement of the incoming tasks within three different contexts, on-line, semi-online, and off-line. In an on-line context, the time for deciding the placement of the incoming tasks is negligible. The optimal placement, seen on the first row of Figure 1, allocates the tasks a_0 and a_1 to the machine m_1, then, a_2 and a_3 to m_2 and finally, a_4 and a_5 to m_3. The run-time of allocated machines is 250 mins. In a semi-online context, incoming tasks are first gathered and then allocated at the start of the next time period. In the example we consider a time period of 3 seconds. Within this context, the optimal placement shown in 2^{nd} row Figure 1 corresponds to 210 minutes total run-time. Machine m_1 allocates its first task at 6 seconds while m_2 and m_3 start respectively at 9 and 12 seconds. In the first time period of 3 seconds the system first collects the tasks a_0, a_1, a_2. From 3 to 6 seconds the system solves the placement concerning the tasks received between 0 and 3 seconds. At the same time, it also collects a_3, a_4, a_5 for future placement. At 6 seconds the system implements last processed solution and allocates a_0 and a_2 to m_0 and a_1 to m_1. From 6 to 9 seconds the system

Figure 1: Optimal placement considering respectively an on-line, a semi-online (time window of 3s), and a off-line contexts.



decides the placement of a_4, a_5. At 9 seconds it implements the solution by allocating a_4 to m_1 and a_3 and a_5 to m_2. Note that in the semi-online framework the waiting time δ between the arrival and the allocation of a task is at most twice greater than the time period of 3 seconds. A small waiting time positively affects on-demand QoS[1].

In the off-line context where all incoming tasks are known before being allocated, the optimal placement shown in 3^{rd} row Figure 1 corresponds to 160 mins of run-time of allocated machines. In this context, the system has decided the placement of the tasks before they arrive, they are allocated as soon as they arrive.

Intuitively, the greater the knowledge of the incoming tasks, the better the consolidation. However, to satisfy on-demand assignment of tasks placement, strategies cannot afford to wait too long to have a greater knowledge. By using a semi-online framework we expect to have more information that can lead to better overall assignments while satisfying on-demand QoS.

3.1. The Semi-Online Framework

We introduce the semi-online on-demand bin packing problem for which the objective is to minimise the global waste of CPU resources allocated across the pool of machines. Table 1 introduces the notations. The semi-online framework [2] is imple-

[1] $*$ = undefined

[2] `https://github.com/ElVinto/semi_online_task_allocation`

Table 1: Notation of the model

Notation	Semantics
i	Time step index
t_i	End time of time step i
tw	Duration of a time step in seconds
$\mathcal{M}$	Set of machines
$\mathcal{M}^i$	State of the machines at time i
C_m	Capacity of machine m
l_m	Maximal expected remaining duration of tasks currently allocated to m
$\mathcal{A}$	Stream of tasks
$\mathcal{A}^i$	Tasks received at step i
t_a	Arrival time of task a
$\bar{t}_a$	Time at which a machine starts processing task a
$\bar{d}_a$	Expected duration of task a in seconds
d_a	Expected remaining duration of task a in seconds
q_a	Requirement of task a
h_i	Map $\mathcal{A}^i \mapsto \mathcal{M}$ from tasks received at step i to allocated machines
z_m^t	Observed run time for machine m between the time points 0 and t
x_{am}	Boolean variable denoting the assignment of task a to machine m
e_m	Auxiliary integer variable e_m denotes the expected runtime of a machine m

mented using two distinct modules as illustrated in Figure 2. The first module acts as a monitor and receives the stream of tasks $\mathcal{A}$ to be allocated to the pool of machines $\mathcal{M}$. At each time step i, from the previous placement solutions sent by the solver, the monitor updates and sends back in the set $\mathcal{M}^i$ the information representing the current state of the machines. From the stream of incoming tasks $\mathcal{A}$, the monitor also gathers the tasks received during the current time step i into the set $\mathcal{A}^i$ before passing them to the solver.

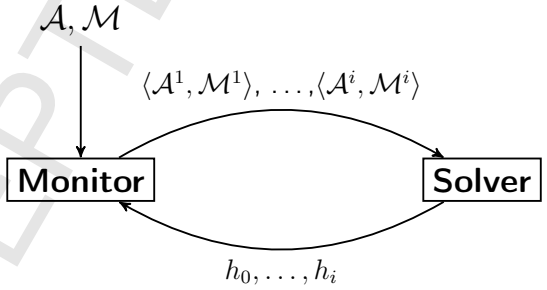


Figure 2: The semi-online framework

Finally from the current state of the machines $\mathcal{M}^i$ and newly arrived tasks $\mathcal{A}^i$ the solver builds the corresponding packing problem. The resulting placement solution h_i, mapping each incoming task to a machine, is then sent to the monitor module.

3.2. The Monitor Module

The monitor module decomposes time into a sequence of time steps of size tw mesured in seconds. Each time step i is mapped to the end time t_i of the corresponding time period. Each task a received during time step i is characterised by an arrival time t_a, an expected duration $\bar{d}_a$ and a required CPU resource q_a. The starting time $\bar{t}_a$ of a corresponds to the end of the next time step after the task has been received, $\bar{t}_a = t_{i+1}$. We implicitly guarantee on-demand placement by requiring the solver to return a consolidation plan within a single time step, which ensures each task will start within 2* tw seconds of its arrival time.

Algorithm 1: Monitor

Input: $\mathcal{A}, \mathcal{M}, h_{i-1}$
Output: $\mathcal{A}^i, \mathcal{M}^i$
1 **for *each*** $a \in \mathcal{A}, t_a < t_i$ **do**
2 **if** $isAlive(a)$ **then**
3 $d_a \leftarrow max(tw, \bar{t}_a + \bar{d}_a - t_i)$
4 **else**
5 $d_a \leftarrow 0$
6 $\mathcal{A}^i \leftarrow \{(a, d_a, q_a) \mid a \in \mathcal{A}, t_i - tw \leq t_a < t_i\}$
7 **for *each*** $m \in \mathcal{M}$ **do**
8 $RunningTasks(m) \leftarrow \{a | \forall j < i, a \in \mathcal{A}^j, h_j(a) = m, d_a > 0\}$
9 $C_m \leftarrow 1 - \sum_{a \in RunningTasks(m)} q_a$
10 $l_m \leftarrow \max(\{d_a \mid a \in RunningTasks(m)\})$
11 $\mathcal{M}^i \leftarrow \{(m, l_m, C_m) \mid m \in \mathcal{M}\}$ return $\mathcal{A}^i, \mathcal{M}^i$

Algorithm 1 provides details on how the *monitor* module is implemented. The monitor first updates (resp. initiates) the remaining expected duration d_a of the tasks received so far (line 1). If a task is still alive ($isAlive(a)$), i.e. a has not completed, its expected remaining duration is updated and corresponds at least to the size of a time step (line 3). Tasks that have completed, have a remaining duration of 0 (line 5). The tasks that have not yet started will have a remaining duration initialized after been scheduled. The monitor then gathers the tasks revealed during time step i into the set $\mathcal{A}^i$ (line 6). For a machine m, the tasks that have been placed but that have not yet completed are gathered into the set $RunningTasks(m)$ (line 8). This set is built upon the previous placement solutions h_j, $j < i$. A placement solution h_j, received during time step j, is a mapping from the incoming tasks A^j to the machines in the cluster $\mathcal{M}$.

From the tasks running a step i, the monitor updates the remaining capacity C_m of each machine m (line 9). It also updates the expected remaining run-time l_m of each machine (line 10). l_m represents the maximal expected duration of tasks currently allocated to the machine m, $l_m = 0$ if the machine is currently hosting no tasks. In the end, the monitor sends both the current state of the machines $\mathcal{M}^i$ and the incoming tasks $\mathcal{A}^i$ to the solver (line 11).

3.3. The Solver Module

At each time step i the solver is called to solve the on-demand bin packing Problem (ODBP) corresponding to the current state of cluster $\mathcal{M}^i$ and newly arrived tasks $\mathcal{A}^i$. The goal is to return a valid placement h_i of the incoming tasks $\mathcal{A}^i$ on the physical servers $\mathcal{M}$ minimizing the expected run-time of allocated machines. The solver has no further knowledge of subsequent arriving tasks. The tasks for which the placement decision is made during time step i will start running on the assigned machines at the beginning of the next time step $i+1$. In the following ODBP problem formulation, each $\{0,1\}$ decision variable x_{am} denotes the assignment of the task a to the machine $m \in \mathcal{M}$ such that $x_{am} = 1$ if task a has been assigned to machine m, $x_{am} = 0$ otherwise. The auxiliary integer variable e_m denotes the expected runtime of a machine m, it is entirely determined by the decision variables x_{am}.

The input data d_a and q_a have the same meaning as before; they represent the expected remaining duration and the CPU requirement of the task a. Similarly, l_m and C_m denote the current maximal expected remaining run-time and the expected remaining capacity of the machine m. We denote by u_m the maximal expected duration of the remaining and the current incoming tasks, $u_m = \max(l_m, \{d_a \mid a \in \mathcal{A}^i\})$. The mathematical model corresponding to the on-demand bin packing problem is as follows: $ODBP(\mathcal{A}^i, \mathcal{M}^i)$:

$$\min \sum_{m \in \mathcal{M}^i} e_m \tag{1}$$

s.t.

$$\sum_{m \in \mathcal{M}} x_{am} = 1 \qquad \forall a \in \mathcal{A}^i \tag{2}$$

$$\sum_{a \in \mathcal{A}^i} x_{am} \cdot q_a \leq C_m \qquad \forall m \in \mathcal{M}^i \tag{3}$$

$$x_{am} \cdot d_a \leq e_m \qquad \forall m \in \mathcal{M}^i \quad \forall a \in \mathcal{A}^i \tag{4}$$

$$x_{am} \in \{0,1\} \qquad \forall m \in \mathcal{M}^i \quad \forall a \in \mathcal{A}^i \tag{5}$$

$$e_m \in [l_m ... u_m] \qquad \forall m \in \mathcal{M}^i \tag{6}$$

At each time step i the solver aims at minimising the sum of the expected remaining run-times of the allocated machines (1). The placement solution returned by the solver enforces the following constraints. Each incoming task is assigned to exactly one machine (2). The sum of CPU requirement of incoming tasks assigned to a machine do not exceed the current machine capacity (3). The expected remaining run time of a machine is bound by the largest remaining duration of any incoming task assign to it (4) and the maximal expected remaining run-time of the machine constraints (6).

A solution of the above mathematical model is a placement reflected in the assignment of the x_{am} Boolean variables (5). Thus, given a current state of the cluster $\mathcal{M}^i$ and newly arrived tasks $\mathcal{A}^i$, the mapping $h_i : \mathcal{A}^i \mapsto \mathcal{M}$ sent back to the monitor is a function s.t. $h_i(a) = m$ if and only if $x_{am} = 1$.

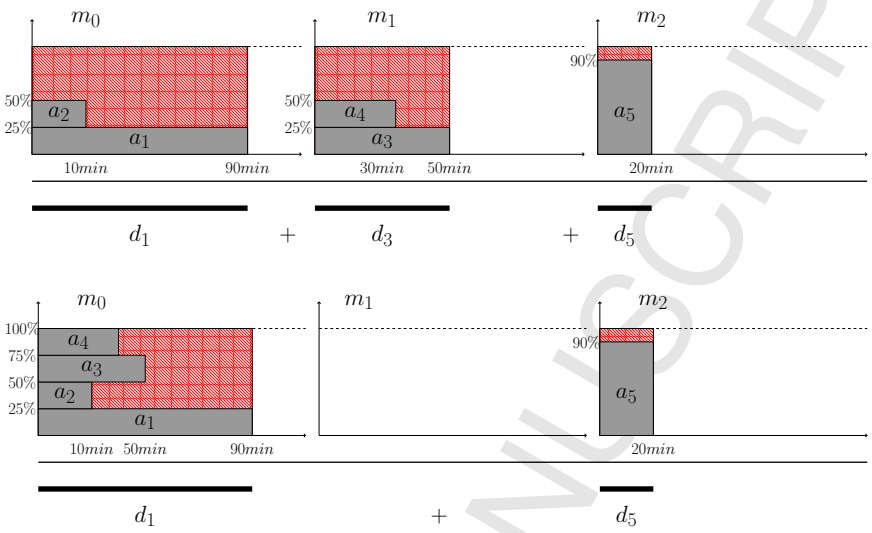


Figure 3: Two valid assignments of tasks $\{a_1, \ldots, a_5\}$ to 3 standby machines $\{m_0, \ldots, m_2\}$. All tasks are starting at the current time step t.

3.4. Illustration of the Consolidation of Run-time of Allocated Machines

In Figure 3, we consider two placements of five tasks, $\{a_1, \ldots, a_5\}$, with the following duration in minutes (min): $d_1 = 90$, $d_2 = 10$, $d_3 = 50$, $d_4 = 30$, $d_5 = 20$. The machines, $\{m_0, m_1, m_2\}$, have the same CPU capacity. The tasks have the following CPU requirements (in percentage of the CPU capacity): $q_1 \ldots q_4 = 25\%$ and $q_5 = 90\%$. In the first assignment depicted at the top of the figure, the objective function evaluates the wastage of m_0 to $w(m_0) = 90 - ((25\% * 10) + (25\% * 90)) = 65$ minutes of CPU resources. m_1 and m_2 waste, respectively, $w(m_1) = 50 - ((25\% * 30) + (25\% * 50)) = 30m$ and $w(m_2) = 20 - (90\% * 20) = 2$ minutes. The total wastage of the first assignment is then $65 + 30 + 90 = 185$ minutes of allocated CPU. Similarly, in the the second assignment of the Figure 3, the total wastage is evaluated to $w(m_0) + w(m_1) + w(m_2) = 45 + 0 + 2 = 47$ minutes of allocated CPU. The second assignment wastes less CPU resources than the first assignment.

Note that the total running time of active machines in the first assignment is $d_1 + d_3 + d_5 = 160$ minutes. The total running time of active machines in the second assignment is $d_1 + d_5 = 110$ minutes. Implicitly, this example gives us insights into the importance of task duration. It also shows the relationship between the total wastage and the total running time of active machines.

Minimising the global duration of active machines is a hard task. In [17] where a static version of ODBP is tackled, the approaches using state-of-the art CPLEX and CP solvers cannot solve the problem optimally for more than a few hundred of tasks. In the context of real world on-demand task placement, heuristic approaches are needed to cope with problem sizes reaching thousands of tasks. In the following, we focus our

study on heuristic methods minimising the wastage in homogeneous platform in data centres for minimising the global duration of active machines.

4. Solution Methods

4.1. Adapting Bin Packing Heuristics

A range of policies originally developed for the bin packing problem can be adapted to produce solutions to ODBP in a semi-online fashion. The hard constraint on the bin capacities is an aspect usually tackled by these policies such as the family of AnyFit (First, Next, Best, Worst) policies, Sum of Squares and Harmonic heuristics.

Due to the semi-online nature of the problem at hand, solving policies must only consider information available up to time i when finding an assignment for tasks $\mathcal{A}^i$. An aspect traditionally not handled by bin packing models is the duration for which an item will be locking resources on its host bin. As shown in Section 3, the current expected run-time of machine m is modeled by l_m.

In addition to the policies themselves, the order of tasks to be assigned in any particular time window may significantly impact the solution's quality. The natural order is given by the **LIST** and leaves the tasks $a \in \mathcal{A}^i$ ordered by their arrival time t_a. Alternatively, the order **D** sorts the incoming tasks by decreasing duration $d_a(i)$. Finally, at every time step i, the set of machines is ordered on their l_m values.

First Fit (FF). The FF policy can be applied as-is to the ODBP problem. The tasks in $\mathcal{A}^i$ are in turn assigned to the first machine that has enough remaining capacity. The condition for assigning task a to machine m at time i is thus $q_a \leq C_m$. Assigning a to m updates the remaining capacity on machine m such that $C_m = C_m - q_a$.

Next Fit (NF). The NF policy maintains a pointer to the machine that was last selected to host a task. In turn, each task $a \in \mathcal{A}^i$ will be assigned to the machine currently referenced by the pointer. If there is insufficient remaining space on this machine (i.e. $q_a > C_m$), NF will move the pointer in to the next machine. If no machine in the list of active machines can accommodate the task under consideration (i.e. the pointer reaches the machine it started with), a non-active machine will host the task. Note that the position of the pointer is maintained across the successive optimisation steps.

Best Fit on Requirements (BFR) / Best Fit on Duration (BFD). Since items are characterised by both size and duration, the best fit policy is ambiguous in the context of ODBP. BFR acts similarly to the Best Fit policy for the bin packing problem. For each task $a \in \mathcal{A}^i$, BF selects $m \in \mathcal{M}$ so that the quantity $C_m - q_a$ is minimised. On the other hand, the BFD policy focuses on finding the machine with the closest running time hence minimising the quantity $|l_m - d_a|$. Naturally if no machine in the set of active machines has enough spare capacity to accommodate the task, a new machine is made active and the task assigned to it.

Max Rest on Requirements (MRR) / Max Rest on Duration (MRD). The Max Rest policy, also known as Worst Fit acts as the opposite of the BFR/BFD policies.

The MRR policy ensures that each task $a \in \mathcal{A}^i$ gets assigned to the machine maximising its unused space: $C_m - q_a$. Similarly, MRD selects for each task the machine maximising the difference in runing time before and after the assignement, hence maximising the quantity $|l_m - d_a|$.

Sum Square - (SS). The Sum-of-Squares algorithm was introduced by János Csirik et al [24]. Sum-of-Squares uses the notion of the gap of a bin which is its spare capacity. The number of bins with spare capacity g is denoted by $N(g)$. Initially, $\forall g : N(g) = 0$. Then SS assigns an item a of size q_a such that the the quantity $\sum_{1 \leq g \leq B} N(g)^2$ is minimised. Here, B stands for the capacity of the bins. The main intuition behind this algorithm is that is maximises the likelihood of finding a item that *almost* perfectly fits the gap in a bin at any time.

Harmonic - (HA). Lee and Lee [25] introduced the Harmonic heuristic for bin packing with the underlying idea being an harmonic partitioning of items and bins on the segment $[0, 1]$ into M families. In our case, the M parameter depends on the number of machines that are hosting tasks at solving time. This partitioning allows us to classify items in an efficient manner. We reuse that idea but instead of partitioning items on their sizes, we partition them on their remaining duration.

4.2. First Merged Fit (FMF)

On-line placement policies (cf. section 4.1) make a clear separation between already allocated machines hosting a set of running tasks and the list of upcoming tasks $\mathcal{A}^i$ that have to be allocated. The policies iteratively allocate tasks to machines, and may miss the opportunity to associate the first two tasks together before assigning them to a machine. This is the main idea behind our algorithm. We propose a flexible approach that does not try to allocate a task to a machine but rather allows each task to be associated to another task or machine.

4.2.1. Illustration

A key concept of the algorithm is to group both allocated machines and arriving tasks under the same notion of *bin*. A bin thus represents either a machine currently running, or a set of tasks to be allocated, or both. In Figure 4, we illustrate an execution of our approach. The first step is to create a bin for each running machine and each arriving task. Then, the list of bins is sorted according to some criteria. In the example, the order of the machines prioritises the longest remaining running time. At each iteration the algorithm merges the best ranked bin with the next compatible bin in the list.

Two bins are said to be compatible if: (i) at most one of the bins is built from a currently running machine, (ii) if one of the bins is active, then merging must be into that bin, without exceeding its capacity; otherwise, merging can be in either bin but must respect that bin's capacity. As an illustration, in Figure 4, b_1 is the bin hosting the longest task. Since the second bin is b_2, b_1 and b_2 are compatible, b_2 and b_1 merge into b_1. The merge operation transfers the tasks from b_2 to b_1 and removes b_2 from the

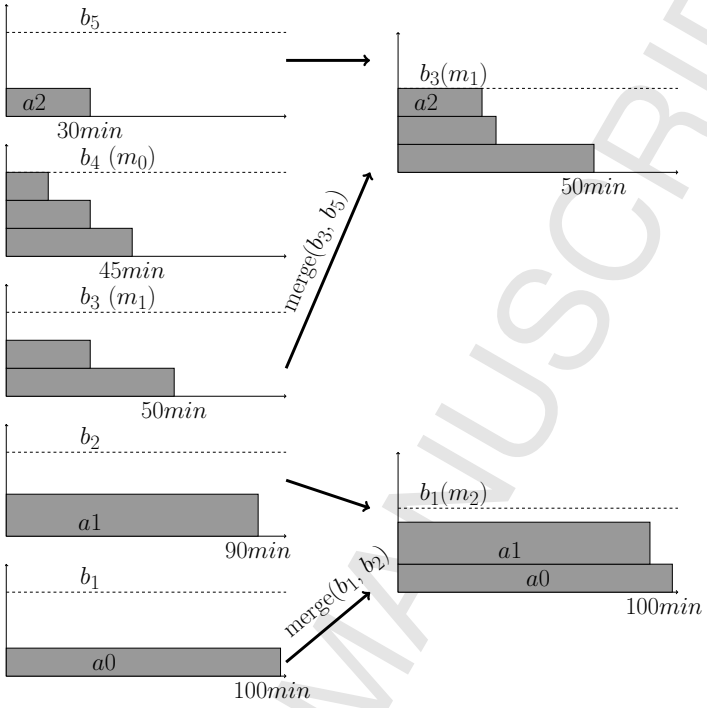


Figure 4: A run of First Merged Fit FMF.

queue. No other bins in the queue are compatible with the updated bin b_1. Tasks from the newly created bin b_1 will be allocated to a new machine (m_2). Next up in the list is b_3 which is built from a currently running machine, as is b_4. Consequently, b_3 and b_4 are not candidates for merging. b_3 will be merged with b_5 to form the updated bin $b_3(m_1)$. The algorithm terminates with no new tasks to be allocated.

4.2.2. Algorithm

Algorithm 2, *First Merged Fit* (FMF), receives the set of incoming tasks $\mathcal{A}^i$ and the current state of the machine $\mathcal{M}^i$ and returns a valid placement h_i assigning the incoming tasks to the machines. First, each machine currently running, and each upcoming task is mapped to a bin (line 1). Implicitly, in our approach, a bin is an object that models the expected state of a physical machine that will be hosting the incoming tasks assigned to it. Thus, a bin either corresponds to an already running machine or represents a machine that will start running with its new set of assigned tasks. The function $buildBins$ returns the bins corresponding to each incoming task and each machine currently running ($C_m < 1$), $buildBins(\mathcal{A}^i, \mathcal{M}^i) = \{(b, d_b, q_b, \mathcal{A}_b) | \forall a \in \mathcal{A}^i, (b, d_b, q_b, \mathcal{A}_b) = (a, d_a, q_a, \emptyset), \forall m \in \mathcal{M}^i, C_m < 1, (b, d_b, q_b \mathcal{A}_b) = (m, d_m, 1 - C_m, \emptyset)$. For a bin b, d_b represents the expected duration of the task, q_b the sum of CPU requirement, $\mathcal{A}_b$ the list of incoming tasks in $\mathcal{A}^i$ newly assigned to m. The list of bins B is sorted according to the maximal duration of tasks of each bin (line 2). Each iter-

Algorithm 2: First Merged Fit (FMF)

Input: $\mathcal{A}^i, \mathcal{M}^i$
Output: h_i
1 $B \leftarrow buildBins(\mathcal{A}^i, \mathcal{M}^i)$
2 $SB \leftarrow sortByMaxDuration(B)$
3 **while** SB *is not empty* **do**
4 $\quad b_i \leftarrow pop(SB)$
5 $\quad b_j \leftarrow nextAllocableWith(b_i, SB, \mathcal{M}^i)$
6 $\quad$ **while** $b_j \neq null$ **do**
7 $\quad\quad b_i \leftarrow merge(b_i, b_j, SB, \mathcal{M}^i)$
8 $\quad\quad b_j \leftarrow nextAllocableWith(b_i, SB, \mathcal{M}^i)$
9 $\quad$ **for** ***each*** $a \in b_i$ **do**
10 $\quad\quad h_i(a) \leftarrow physicalMachine(b_i)$
11 return h_i

Algorithm 3: merge

Input: $b_i, b_j, SB, \mathcal{M}^i$
Output: b_r
1 $SB \leftarrow SB \setminus b_j$
2 **if** $b_i \in M^i$ **then**
3 $\quad b_r \leftarrow b_i$
4 $\quad b_s \leftarrow b_j$
5 **else**
6 $\quad b_r \leftarrow b_j$
7 $\quad b_s \leftarrow b_i$
8 $d_{b_r} \leftarrow max(d_{b_r}, d_{b_s})$
9 $q_{b_r} \leftarrow q_{b_r} + q_{b_s}$
10 $\mathcal{A}_{b_r} \leftarrow \mathcal{A}_{b_r} \cup \mathcal{A}_{b_s}$
11 return b_r

Algorithm 4: $nextAllocableWith$

Input: $b_i, SB, \mathcal{M}^i$
Output: b_j
1 **for** ***each*** $b_j \in SB$ *from j* **do**
2 $\quad$ **if** ***not*** $(b_i \in \mathcal{M}^i$ *and* $b_j \in \mathcal{M}^i)$ **then**
3 $\quad\quad$ **if** $q_{b_i} + q_{b_j} \leq 1$ **then**
4 $\quad\quad\quad$ return b_j
5 Return null

ation sees the best-ranked bin b_i, line 4, merged with the next compatible bin b_j in the queue (line 7). When the bin is filled, i.e. no further bins can be merged with the current bin, each task allocated to the bin is then map to a physical machine for placement. The function $physicalMachine(b)$ returns either the running machine corresponding to the bin or an unassigned machine.

Algorithm 3 gathers the tasks of two bins in one (line 10) and updates the state of the bin accordingly (lines 8-9). In the case of a merge between a bin associated with a running machine and a bin associated with a new machine, the bin receiving the merge is the bin associated with the running machine (lines 3-6).

Algorithm 4 searches the next compatible bin starting from the index of the last visited bin j and iterates over the queue of unvisited bins (line 1). If a compatible bin is found, i.e. the input bin b_i and the visited bin b_j do not both represent running machines (line 2) and the sum of the resources requirement is not excessive (line 9), the two bins will be merged (cf. Algorithm 3). If no compatible bin is found, the bin is completed, its new allocated tasks will be placed and start running on the corresponding machine at the next time step.

Let $n = |B|$ be the number of bins, i.e. the number of running machine plus the number of upcoming tasks. The complexity of sorting the list of bins is $O(nlog(n))$.

For each bin in the list (Algorithm 2 Line 3) we only iterate over the remaining part of the list (Algorithm 4) in descending order. In the worst-case scenario, no compatible bins are found, so the list of bins does not decrease over the iteration. In this case the complexity is $n * (n-1)/2$. Overall, the complexity of FMF is $O(nlog(n) + n * (n-1)/2) = O(n^2)$.

4.3. Enhancing Policies with Local Search

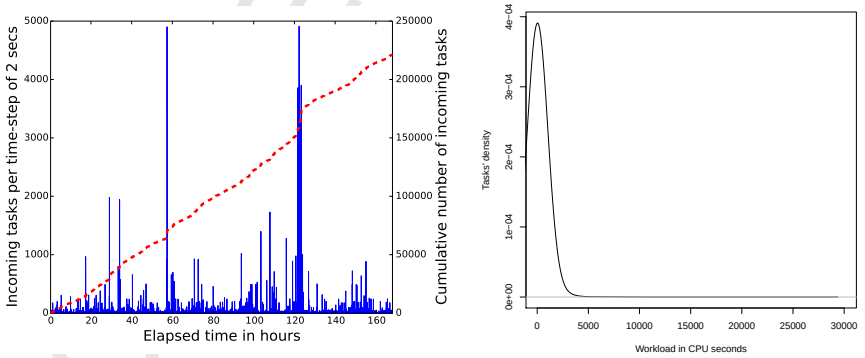
The previously described policies can be used to produce valid solutions heuristically in a rather short amount of computational time. In a real operational setting one could use the time left in the window to try to converge to better solutions using techniques such as local search. The underlying idea is to build a MIP model capturing the decision problem local to a time window. This local problem is composed of the state of the system and the list of incoming tasks for which an assignment is expected. The various policies are used to produce a feasible solution (incumbent) in turn provided as a first assignment to the complete solver (CPLEX). The complete solver is then allowed the time left to explore further solutions.

CPLEX was tuned to use a local search technique (RINS) method described in [26] as a black box. RINS exploits information contained in the linear relaxation of the MIP model. RINS can be thought as an anytime approach to local search in the sense that it always yields the best feasible solution found so far.

5. Empirical Evaluation

5.1. Dataset

We evaluated the different approaches against a data-centre trace released by Google. From the trace we extracted the information of one week's worth of tasks.



(a) Number of incoming tasks over elapsed time

(b) Empirical distribution of tasks over the workload

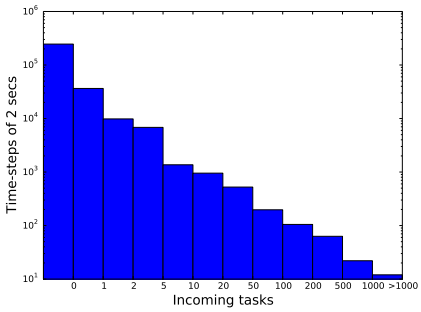
Figure 5: Characterizing incoming times and workload distribution of tasks in the dataset.

Figure 5a shows a high variability in the number of incoming tasks received over time. Each vertical bar represents the number of incoming tasks (left y-axis) received within a time period of 2 seconds at a specific time (x-axis). The cumulative number of incoming tasks (right y-axis) is depicted by the red dotted line. The dataset shows that the number of tasks received by two consecutive time steps can differ by several orders of magnitude. There are two noticeable activity peaks of around 5000 tasks arriving after 57 hours and 123 hours. The second noticeable peak is closely surrounded by other peaks of several thousand tasks. This specificity in the dataset is of particular interest since it allows us to compare the behavior of the different policies in case of intensive demand of resources in Section 5.2.2.

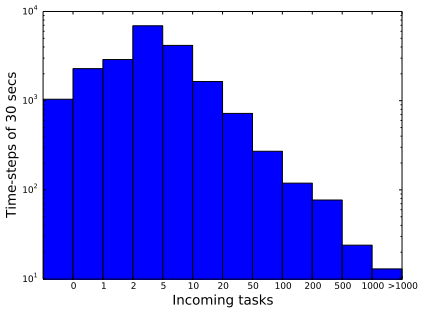
Figure 5b represents the tasks empirical distribution of workload. The workload of a task is computed by multiplying its CPU-requirement by its duration. CPU-requirement of tasks is expressed in terms of the percentage the largest capacity server in the data-centre. The figure shows that the dataset is extremely skewed toward small tasks. There is a rather wide gap between small tasks (e.g. a few CPU seconds) against large tasks (e.g. thousands of CPU seconds). We discuss in more detail of the dataset's heavy tail behavior when introducing figure 6c and Figure 6d.

Figure 6 shows various task distributions of the dataset according to different parameters. Figure 6a, (resp. Figure 6b) shows the number of time steps of 2s, resp. 30s, when varying the interval of tasks received within a time step. Over the week of incoming tasks, while considering 2s time period, more than 246000 time steps of 2s representing a sum of 56 hours receive no tasks (Figure 6a), while only 1038 time steps of 30s representing a sum of 21 hours are empty (Figure 6b). The difference between the number of time steps of 2s and 30s remains significantly high when 1 or 2 tasks are received within a time step. Then, as expected, since tasks have more chance to be pooled within time steps of longer time periods, there are more time steps of 30s that receive between 6 to 10 tasks, 11 to 20 tasks, ..., > 1000 tasks than the number of time steps of 2s. Note that, for both the 2s and 30s cases, there are more than 700 time-steps receiving between 20 and 100 tasks. Finding an optimal placement for these time steps remains challenging even for state of the art techniques and solver [17]. From more than 100 tasks received during a time step, the difference between the number of time steps of 30s and 2s is tightening and becomes almost equal. There are 13 time steps of 30s against 12 time steps of 2s that received more than 1000 tasks. This pattern is specific to the dataset built from a real trace of incoming tasks. As shown in Figure 5a peaks of incoming tasks are spread across time. Implicitly, in case of peaks of demand, the associated placement problems are very hard to solve optimally within a service level agreement matching on-demand QoS expectations. Heuristics and semi-online policies are required in this kind of applications.

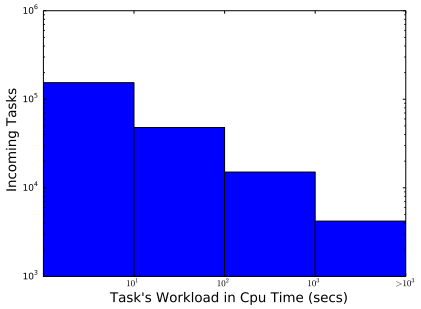
Note that, For 2s and 30s, the number of time steps decreases logarithmically as the number of incoming tasks increases. As a consequence for both time periods 2s and 30s most of the time steps received less than 10 incoming tasks. As shown Figure 7, when there are few tasks per time step, there are no noticeable differences between the placement policies. Differences are more noticeable at the arrival of peaks of tasks, i.e., when the placement problem become harder to solve. On the other hand, because of the sparse distribution of peaks of tasks over time (> 30s) and the abundance of under loaded time steps, both the number of tasks per peak and the number of peaks



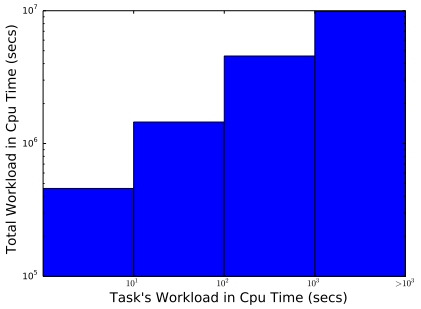
(a) Number of time steps of 2s (y-axis log scale) that received $(x, x + \delta]$ tasks (x-axis)



(b) Number of time steps of 30s (y-axis log scale) that received $(x, x + \delta]$ tasks (x-axis)



(c) Number of incoming tasks (y-axis log scale) having a workload in between $(x, x+\delta]$ Cpu Time secs (x-axis log scale)



(d) Total Workload of Tasks (y-axis log scale) having a workload in between $(x, x + \delta]$ Cpu Time secs (x-axis log scale)

Figure 6: Distributions of incoming tasks in the dataset

remain close for 2s and 30s time periods. As a consequence, the placement problem is almost as hard to solve for 2s and 30s time periods, see Figure 9, there is only a small difference in overall consolidation quality.

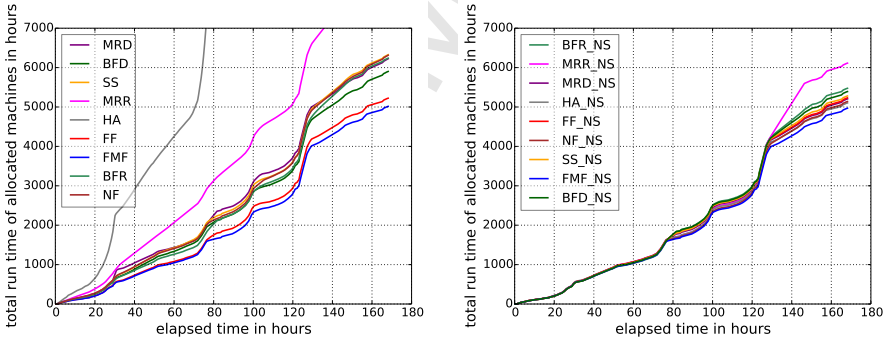
The dataset contains 221.140 tasks representing a total workload of 16.343.165 CPU seconds. Figure 6c and Figure 6d respectively show the distribution of incoming tasks and the total workload of tasks over an interval of individual task workload. Figure 6c shows that the dataset contains 153.879 small tasks, $\leq$10 CPU seconds, representing almost 70% of the incoming tasks, while it only contains 4207 large tasks representing only 1.9% of the incoming tasks. On the contrary, Figure 6d shows that the rather small tasks, $\leq$1000 CPU seconds, represent approximatively 40% of the total workload, while the largest tasks, $>$1000 CPU seconds, represent 60% of the workload. These last two distributions are representative of the heavy tail behavior of real world workload distribution in data-centre where a majority of small tasks corresponds a roughly half of the total workload and a smaller proportion of larger tasks represents the other half.

5.2. Results of the Experiments

In theses experiments, we first compare the performance of the workload consolidation of the incoming tasks as reported in the trace (i.e. we assume we know the actual duration of each task when it arrives). Section 5.2.1 compares the total allocated resources of the different approaches for the period one week of incoming tasks. Section 5.2.2 analyses the resource consumption of the policies during periods of high activity. Section 5.2.3 measures the resource usage when varying the time step durations. Section 5.2.4 relaxes the hypothesis on full knowledge of tasks' duration and measures the resource usage when varying the uncertainty of tasks' duration. Section 5.2.5 compares the time performance. The last section 5.2.6 summarises the resource usage and the time performance of the approaches.

5.2.1. Total Resource allocated and use of Neighbourhood Search Heuristics

Figure 7 shows the solution quality of the different approaches by comparing the total allocated resources over time. The allocated resource is measured in term of runtime of allocated machines. The time step duration is two seconds. Figure 7 a) shows the evolution of the total resource usage of each policy. Figure 7 b) shows the evolution of the total resource usage when the remaining time left is used to improve the solution returned by each policy using a neighborhood search heuristic [27] with the CPLEX solver.



a) Policies b) Policies + Neighbourhood Search

Figure 7: Total allocated resources over the time per policies against elapsed time

The total resource allocated over time interleaves steady growths and steps. Steady growth corresponds to time periods when few tasks arrived per time-window. Steps correspond to arrivals of peaks of incoming tasks. In periods of steady growth all the approaches slightly increase the total allocated resources. The gap in resource utilisation of the approaches slightly increases after each change of step.

At these times, the placement of tasks becomes more challenging and the difference between the solution quality of the policies increase accordingly. We analyse in details the arrival of peaks of tasks in Section 5.2.2.

The best placements policies are FMF followed by FF then BFD. The HA and MRR policies waste significantly more resources and exceed the resource limit shown

in the Figure 7 a). When the solution of each policy is enhanced by the neighbourhood search, Figure 7 b), less efficient policies significantly improve their resource usage and narrow the gap between the best policies. Note that the use of local search marginally improves the resource usage of the best policies.

Table 2: Total run-time of allocated machines (in hours) of the placement policies

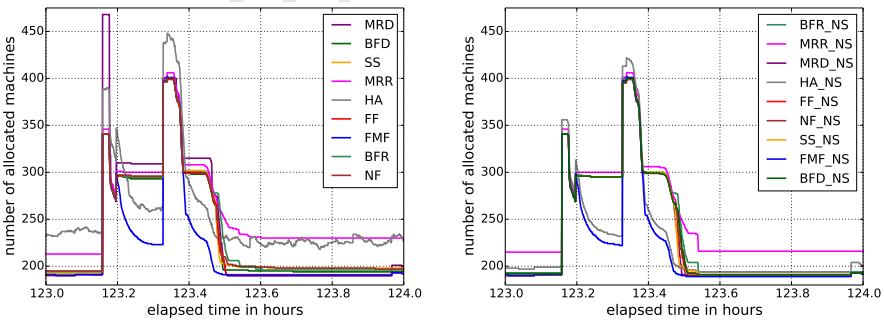
Approaches:	FMF	BFD	BFR	FF	HA	MRD	MRR	NF	SS
Policy:	5016	5902	6239	5223	18232	6225	8786	6313	6327
Policy+NS:	4969	5392	5474	5231	5102	5205	6113	5138	5279

In Table 2 we present the total resources allocated by the different approaches after seven days, i.e., 168 hours of workload consolidation. Compared to the leading standard bin-packing heuristics, FMF consumes 4% less resource than FF and 15% less resource than BFD. Compared with the most inefficient policies, FMF saves 1.75 and 3.6 times more resource than MRR and HA.

5.2.2. Extra Resource Allocated and Released during Peaks of Incoming Tasks

We analyse the resources allocated and released after incoming peaks of tasks. Figures 8 a) and 8 b) compare the behaviour of the approaches at the arrival of three peaks of 3293, 959 and 3899 tasks received at different time steps. During the 123^{rd} hour 10246 tasks have to be placed, the three noticeable peaks represent 80% of the incoming tasks. In both figures, the x axis corresponds to a specific time span of one hour, i.e. 1800 time steps of 2 seconds of the 123^{rd} hour of the simulation of task arrivals. The y axis describes the number of allocated machines.

Figure 8: Number of allocated machines over the peaks of the 123^{rd} hour



a) Policies

b) Policies + Neighbourhood Search

In Figure 8 a), at the arrival of the first noticeable peak of tasks, all the approaches perform similarly. They allocate new machines to cover the peak of activity. The same behaviour can be observed at the arrival of the second and the third noticeable peaks. The quality of the tasks consolidation methods can in fact be observed after the peaks when the CPU resource is gradually released by the tasks finishing their execution.

Placement policies from the related work waste significantly more resource than FMF. At the end of the first and the third peaks, HA allocates up to 40 more machines than FMF while the others placement policies allocate up to 80 more machines than FMF.

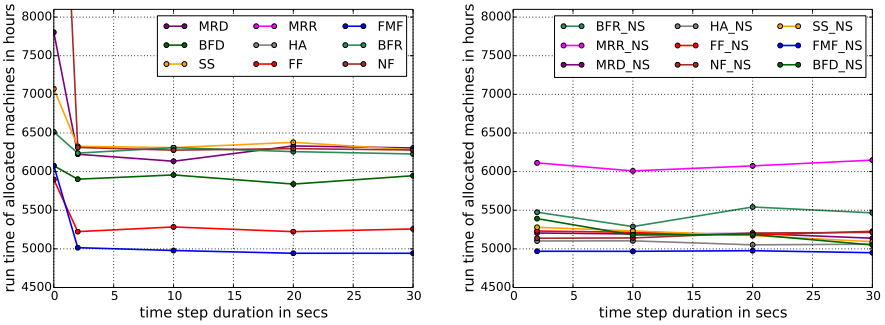
Figure 8 b) shows the resource allocated by the policies enhanced by neighbourhood search heuristics. Only the less efficient placement policies HA, MRD and NF show a clear benefit of using the neighbourhood search heuristic to improve their solutions. At the arrival of the first peak of incoming tasks the number of allocated machines used by MRD is significantly reduced when using local search and drops from more than 450 machines to less than 350 machines. The resource consumption of HA enhanced by neighbourhood search is also significantly reduced. It remains constantly closed to FMF along the hour. For the other methods, the local search approach does not improve the solution returned by the policies. In these cases, the solutions returned by the policies are already good. The local search does not have the time to improve the placement within the time step of two seconds. In Table 3, 4^{th} line, we also check the performance of the enhanced policies when the duration of each time step has been increased to 30 seconds. The results show that with longer time step duration, the gain in resources utilisation is not significant for the policies already showing an efficient consolidation.

Table 3: Run-time of allocated machines above 180 machines during the 123^{rd} hour (in hours)

	Policy:	FMF	BFD	BFR	FF	HA	MRD	MRR	NF	SS
tw: 2s	Alone	35.8	51.8	54.9	49.7	74.7	57.2	76.8	55	54.1
	+NS	35.3	51	53.5	50	43.2	49.5	69.9	50.6	50.3
tw: 30s	Alone	35.9	52.4	52.2	48.6	71	53.5	75.5	53.2	53.4
	+NS	36.4	48.4	50.2	48.1	42.1	48.5	67	48.6	48.8

Note that at the beginning and at the end of the 123^{rd} hour more than 180 machines remain allocated. Table 3, shows the extra resource above 180 machines allocated during this hour. In the first part of Table 3, we show the extra resources allocated when the time step is 2 seconds. In order to accommodate the peaks of incoming tasks of the 123^{rd} hour, FF allocates 40% more resources than FMF. The other policies, such as BFD and HA, allocate between 40% and 2 times more resources than FMF. Table 3, 2^{nd} line, shows that HA enhanced by neighbourhood search approach drastically reduces the resource consumption by 42%. MRD and NF are improved by approximatively 8%. FMF, BFD, BFR and FF show similar resources consumption enhanced or not by neighbourhood search. In these cases, the consolidation is already efficient, the remaining time dedicated to the local search heuristic to improve the polices' solutions shows no benefit. Note that HA enhanced by a local search heuristic returned better results than BFD, BFR, FF. Implicitly it seems that the local minimum found by the local search starting from HA solutions is better than the local minimum found when starting from BFD, BFR and FF solutions.

In the second part of Table 3, we show the extra resources allocated when the



a) Policies

b) Policies + Neighbourhood Search

Figure 9: Allocated resources per policies when increasing time step duration

time step is 30 seconds. In this case all the policies slightly improves their resource consumption except FMF, BFD, FF, showing similar results to the ones obtained earlier. Using local search heuristic drastically improves the allocated resources of HA and more reasonably the resources consumption of MRR, NF, SS, BFD, and BFR. FMF and FF return similar resource consumption than before.

In summary, during a period of high resource demand, the placement policies may pay the price of an inefficient placement after the peaks. As a result when the short duration tasks ended an unnecessary amount of machines has remained active to execute longer time duration tasks. This can be the result of a suboptimal placement of tasks at decision time. FMF is showing a more efficient usage of the allocated resource. This behaviour can be explained by the fact that FMF takes full advantage of the tasks ordering based on duration compare to some other policies such as BFD or MRD. In addition FMF exploits the opportunity to merge incoming tasks before allocating them to a machine.

5.2.3. Total Resource Allocated When Varying Time Step Durations

Figure 9 shows the evolution of the resource usage of the different approaches when increasing the time step duration. The resource usage (y-axis) is expressed in run time hours of allocated machines. The time step durations (x-axis) is expressed in seconds. Note that increasing the time step duration implicitly decreases on-demand QoS ensuring real time placement of tasks. In an on-line context each task as to be placed as soon as it arrives in the system. In the experiments we simulate the on-line context by considering an ordering heuristic based on the arrival time of the tasks. The result of the on-line simulation is shown by the time step duration 0s. The benefit of the semi on-line context described in this study is shown from the time step durations 2s to 30s. We consider 30 seconds to be the upper bound on acceptable QoS level.

In Figure 9 a) each policy shows a noticeable improvement of the resource usage from the on-line context, i.e. a time step duration of 0s, to the semi-online context, i.e. a time step duration of 2s. Within the semi on-line context, i.e. from a time step of 2s to a time step of 30s, the resource usage slightly decreases or remains stable as the time step duration increases. FMF always shows the best resource utilisation

when increasing the time step duration, and is closely followed by FF. Figure 9 b) shows the evolution of the resource usage of the placement policies enhanced by the neighbourhood search. Compared to the solutions returned by policies, the policies enhanced by neighbourhood search show a noticeable improvement in the resource usage. FMF and FF are the exceptions, the resource usage remain stable.

Table 4 sums up the total resource usage at the specific time duration of 0s, 2s and 30s. In the online context (time step duration = 0), FMF is similar to BFD. From time step duration 0s to 2s, NF, MRD and FMF respectively reduce the allocated resources by 50%, 20% and 17%. These improvements represent the best gains among the policies for this variation of time step duration. Then, only HA shows a significant decrease in the resource consumption and allocates 22% less resources from a time step duration of 2 to 30 seconds. FMF only reduces it allocated resources by 1.5%. When it comes to the policies enhanced by neighbourhood search, only BFD shows a slight decrease in the resource utilisation of 6%. The other enhanced policies keep similar amounts of allocated resources. Enhanced policies such as MRR+NS, NF+NS slightly increase the resource consumption when passing to a time step duration of 30s. These cases are counter-intuitive since the same methods give better results in a context with less available information. In the general case, solving a succession of locally optimized problem leads closer to the global optimal. However it is not always guaranteed. In our experiments the cases of MRR+NS and NF+NS remain marginal.

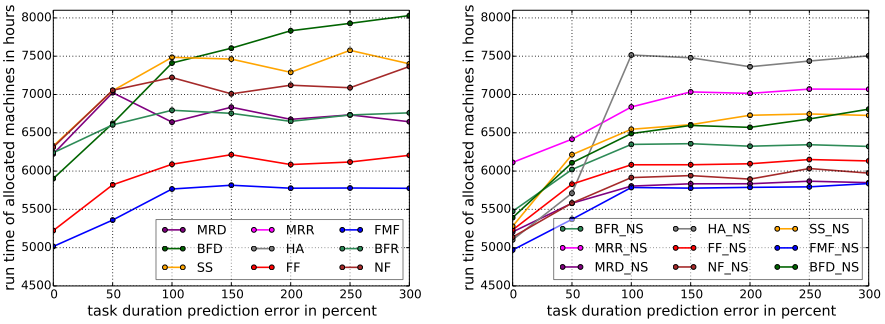
Table 4: Resource utilisation (in hours) for time step durations 0, 2 and 30 seconds

Policy	tw	FMF	BFD	BFR	FF	HA	MRD	MRR	NF	SS
Alone	0*(s)	6076	6076	6514	5898	18493	7803	9643	13032	7072
	2(s)	5016	5902	6239	5223	18232	6225	8786	6313	6327
	30(s)	4943	5947	6228	5258	14255	6306	8648	6280	6280
+NS	2(s)	4969	5392	5474	5231	5102	5205	6113	5138	5279
	30(s)	4950	5050	5466	5228	5061	5138	6149	5213	5093

5.2.4. Total Resources Allocated When Varying Task Duration Uncertainty

In this section we relax the hypothesis on full knowledge of duration and measure resource usage when varying the uncertainty of task durations. In our experiments, given a task duration prediction error of $x\%$ and a task duration d recorded from the data centre traces, the policy is only given the expected task duration in $[max(0, d.(1 - x/100)), d.(1 + x/100)]$ to decide on its placement. As shown in Algorithm 1, the monitor updates the expected remaining duration of tasks as the time passes. A task is effectively deallocated only when its real duration ends, i.e. the task has completed or has been interrupted. Note that a duration prediction error of 100% means that the expected duration is randomly chosen between 0 and twice the real task duration. A duration prediction error of 300% means that the expected duration is randomly chosen between 0 and four times the real task duration.

Figure 10 a) and b) show for each policy the evolution of the allocated resource when increasing the task duration prediction error. Both type of approaches, i.e., poli-



a) Policies

b) Policies + Neighbourhood Search

Figure 10: Allocated resources while increasing tasks duration uncertainty

cies Figure 10 a) and policies enhanced by neighbourhood search Figure 10 b), follow the same pattern. First, the resources allocated by the approaches increase linearly as the error prediction increase from 0% to 100%. Then, from 100% the resources allocated remain constant or slightly increase. Within the approaches based on policies, FMF is noticeably better and shows better consolidation. The second best is FF while the other policies consume significantly more resources. The policies HA and MRR are outside the scope of the figure, their performance are shown in Table 5. Within the approaches using neighbourhood search, FMF+NS remains better for prediction error between 0% and 100%. Then, it is closely followed by MRD+NS. Here MRD takes a clear advantage of the neighbourhood search heuristic to improve its resource consumption. It is also the case for HA+NS and MRR+NS that now appear in the scope of the figure.

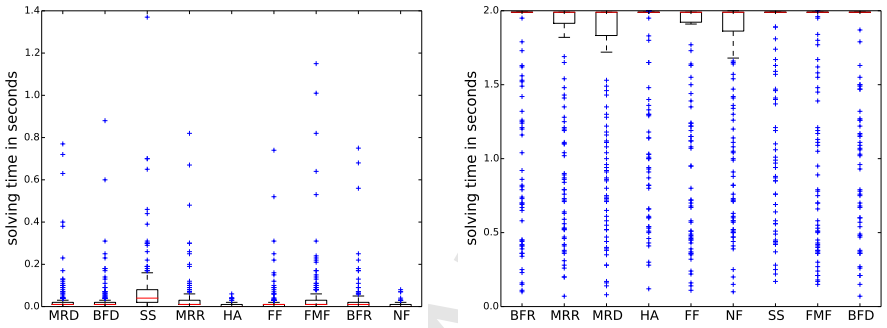
Table 5: Total run-time of allocated machines (in hours) of the placement policies tw=2

Policy	err	FMF	BFD	BFR	FF	HA	MRD	MRR	NF	SS
Alone	0%	5016	5902	6239	5223	18232	6225	8786	6313	6327
	100%	5766	7411	6794	6089	21181	6639	10590	7222	7485
	300%	5775	8031	6760	6206	21271	6644	10420	7366	7401
+NS	0%	4969	5392	5474	5231	5102	5205	6113	5138	5279
	100%	5786	6490	6348	6082	7515	5804	6837	5915	6546
	300%	5837	6809	6322	6133	7505	5849	7068	5973	6727

Table 5 sums up the two patterns followed by the approaches when increasing the task duration prediction error. For a prediction error of 100%, the policy FMF wastes 5% less resources than FF and between 13% and 23% less than the other policies. MRR and HA for that consumes 1.8 and 3.7 times more resources than FMF. For a prediction error of 300%, the policy FMF wastes 7% less resources than FF and between 13% and 28% less than the other policies. Enhanced policies using neighbourhood search drastically reduce the resource usage of all policies except FMF+NS and FF+NS.

In summary we have analysed the behaviour of the different approaches when increasing the uncertainty of the task duration prediction error. Here again enhancing FMF or FF by a neighbourhood search does not improve the resource consumption, and confirms the quality of the the solution returned by these policies. The other approaches clearly benefit from a local search and reduce the gap with FMF.

5.2.5. Real-time Placement of Incoming Tasks



a) Policies b) Policies + Neighbourhood Search (NS)

Figure 11: Solving time (in seconds) of the placement policies when number of incoming taks>100

Figure 11 compares the solving time quartiles of the different policies (Figure 11 a)) and the policies enhanced by Neighbourhood search (Figure 11 b)). Here we consider only the solving time for placement problems having more than 100 incoming tasks during a time-step during of 2 seconds. We show, for each approach, the solving time quartiles using a candlestick. In Figure 11 a), the approaches based on policies only show an almost instantaneous median solving time that does not exceed 0.1 second. The highest point at the top of each policy denotes the time for solving the largest peak of thousands tasks. The largest peak is solved in 1.2s and 1.4s for FMF and SS, and less than 1 seconds for other methods. NF always answers instantaneously. The NF policy only pays the cost of sorting the task by duration and then assign them as they come to the next available machine. Even if FMF is able to allocate new incoming tasks into a new machines without the need to iterate over all the allocated machines, it pays the price of rearranging in one list both the incoming tasks and the currently allocated machines. Nevertheless, since the worst solving time remains below the time step duration and FMF is able to satisfy the on-demand QoS enforce by the semi-online framework. Contrastingly, in Figure 11 a), the approaches based on policies plus neighbourhood search hit the time limit of 2 seconds corresponding to the time step duration. Implicitly, the solver driven by a neighbourhood search tries to improve the given policy solution till the end without proving the optimality. As noticed before, increasing the time step duration from 2 secs to 30 secs drastically improves the solution found by less efficient policies but does not improve the resource usage of FMF. Consequently FMF+NS pays an unnecessary computation time.

In summary, with a time step period of 2s, all the heuristics are able to place the incoming tasks instantaneously and within the time limit. Only NF Heuristic shows

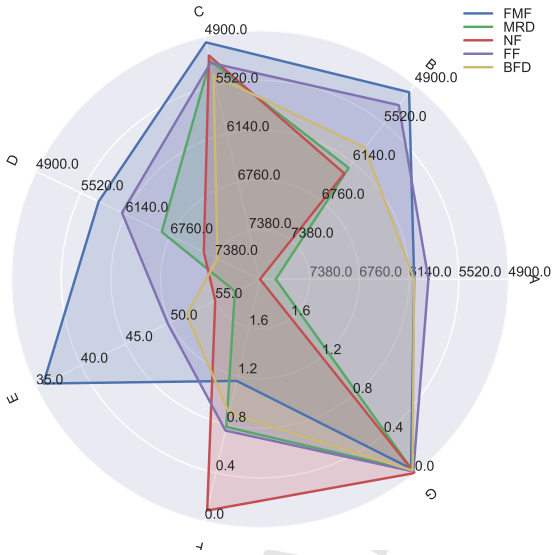


Figure 12: Performance of the policies while varying different parameters

instantaneous placement time in period of peaks of incoming tasks. In this case FMF shows the 2^{nd} worst placement time but remains below the time limit. This modest time performance is compensated by a better resource usage.

5.2.6. Summary of the Resource Usage and the Time Performance of the Policies

In this section we summarise the performance of the FMF, FF, BFD, MRD and NF policies through the radar plot shown in Figure 12. The performances of the policies are compared in terms of resource utilisation (axes A, B, C, D, E) and time performance (axes F, G). For a given policy, the closer to the border, the better the policy is performing. At a glance, we can see that FMF is the best or equal best policy on all axes except one the solving time. Nonetheless FMF still shows solving time satisfying On-Demand placement QoS within 2 seconds.

Axis A represents the total resource utilisation of the the policies in the on-line context (cf. section 5.2.3). In this context FF shows the best resource usage. It is closely followed by FMF and BFD.

Axis B shows the benefit of the semi-online context. Compared with the on-line context it guarantees a placement within a 2 seconds time-step. In this case, all policies noticeably improve their resource usage compared with the on-line context. FMF becomes the best policy and shows an improvement of 17% of its resource utilisation. Even if NF shows the less efficient consolidation it is this policy that benefits most from the semi-online context and wastes 50% less resources. As discussed in Section 5.2.3, from a time step of 2 seconds to a time step of 30 seconds, the policies do not improve significantly their solution quality.

Axis C shows the benefit of enhancing the policies with a neighbourhood search heuristic in the semi-online context of 2 seconds time-step. Compared to policies on

their own, the benefit of local search is more noticeable for MRD and NF the less efficient policies that strengthen the gap between FMF. FMF shows the best resources utilisation but it only improves its resource utilisation by 1.5%. This improvement does not sensibly change when more time is dedicated to the neighbourhood search (cf. section 5.2.3). This is due to the dataset where peaks of incoming are spread over time. This also confirms the quality of the solutions returned by FMF.

Axis D represents the total resource utilisation in a semi-online context with 2 seconds time-step and an uncertainty of task duration of 100%. Due to the uncertainty of task duration all policies become less efficient and waste more resource. However, FMF shows the best resource consolidation even if it degrades its resource utilisation by 15%. More importantly the waste is limited and remains stable as the uncertainty in the task duration goes from 100% to 300% (cf. section 5.2.4).

Axis E represents the extra resource consumes during the 123^{rd} hour at peaks of incoming tasks. This time period represents a more challenging placement problem since the policies have to deal with large amount of tasks in a short time. Here, FMF clearly outperforms the other policies. The second best policy allocates up to 40% more resource than FMF (cf. section 5.2.2).

The last axes show respectively the maximal solving time (F) and the median solving time (G) for time step receiving more than 100 incoming tasks. The maximal solving time corresponds to the placement of a peak of thousands tasks. Here NF answers almost instantaneously. FMF show the slower solving time even if it remains below 2 seconds (cf. axis F). In the average case all policies are qualified as real-time approaches (cf axis G and section 5.2.5).

6. Conclusion

Because of their economic and environmental footprint, workload management in data centres is an important challenge. Workload consolidation is one way to reduce the wastage of resources by clustering tasks together on a subset of the pool of available machines. In the literature many successful approaches have studied the problem of task consolidation from different perspectives. In this study, we tackle the challenge of task consolidation in the context of on-demand resource allocation where data centres want to guarantee real-time allocations of users' tasks. While most of the approaches have envisaged on-line consolidation policies, placing one task at a time, or batch consolidation optimisation, allocating the tasks off-line, we consider the workload consolidation in the context of semi on-line optimisation. In this new context, we introduce a novel approach that benefits from the short period time windows to take more informed decisions while satisfying real-time requirement of on-demand placement QoS.

We have introduced a model allowing us to reason about the dynamics of the problem. We presented bin packing heuristics (FF, NF and BF) along with our ad-hoc algorithm (FMF) that implement semi-online workload consolidation by locally (in time) minimising resource wastage.

We have seen that FMF outperforms bin-packing inspired heuristics on the problem as we have formulated it. After one week of workload consolidation, FMF saves up to 40% more resources during periods of high resources demand than the best adapted

heuristics enhanced with local search. The gap between FMF and the other approaches is more visible in period of peaks of incoming tasks. Moreover, FMF also shows the best resource utilisation performance when increasing the uncertainty of task durations or varying the time period of time step. Even if a better resource usage comes at the cost of a more time consuming approach, FMF is able to guarantee on-demand QoS. In future work, we will focus on exploiting improved measures of runtime duration for the incoming tasks, and on opportunities for workload balancing between data centres.

Acknowledgments

This research has been funded in part by Science Foundation Ireland (SFI) under Grant Number SFI/12/RC/2289.

References

[1] A. Shehabi, S. Smith, D. Sartor, R. Brown, M. Herrlin, J. Koomey, E. Masanet, N. Horner, I. Azevedo, W. Lintner, United states data center energy usage report.

[2] A. Greenberg, J. Hamilton, D. A. Maltz, P. Patel, The cost of a cloud: Research problems in data center networks, SIGCOMM Comput. Commun. Rev. 39 (1) (2008) 68–73. doi:10.1145/1496091.1496103.
URL http://doi.acm.org/10.1145/1496091.1496103

[3] A. Beloglazov, R. Buyya, Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers, Concurr. Comput. : Pract. Exper. 24 (13) (2012) 1397–1420. doi:10.1002/cpe.1867.

[4] J. G. Koomey, Worldwide electricity used in data centers, Environmental Research Letters 3 (3) (2008) 034008 (8pp).
URL http://stacks.iop.org/1748-9326/3/034008

[5] R. C. Hemanandhini I.G., A survey on vm consolidation for energy efficient green cloud computing, International Journal of Emerging Technology in Computer Science and Electronics. 19.

[6] F. L. Pires, B. Barán, Virtual machine placement literature review, CoRR abs/1506.01509.
URL http://arxiv.org/abs/1506.01509

[7] R. W. Ahmad, A. Gani, S. H. A. Hamid, M. Shiraz, A. Yousafzai, F. Xia, A survey on virtual machine migration and server consolidation frameworks for cloud data centers, J. Network and Computer Applications 52 (2015) 11–25. doi:10.1016/j.jnca.2015.02.002.
URL http://dx.doi.org/10.1016/j.jnca.2015.02.002

[8] A. Wolke, B. Tsend-Ayush, C. Pfeiffer, M. Bichler, More than bin packing: Dynamic resource allocation strategies in cloud data centers, Information Systems 52 (2015) 83 – 95, special Issue on Selected Papers from {SISAP} 2013. doi:http://dx.doi.org/10.1016/j.is.2015.03.003.
URL http://www.sciencedirect.com/science/article/pii/S0306437915000472

[9] T. C. Ferreto, M. A. Netto, R. N. Calheiros, C. A. D. Rose, Server consolidation with migration control for virtualized data centers, Future Generation Computer Systems 27 (8) (2011) 1027 – 1034. doi:http://dx.doi.org/10.1016/j.future.2011.04.016.
URL http://www.sciencedirect.com/science/article/pii/S0167739X11000677

[10] J. O. Iglesias, M. D. Cauwer, D. Mehta, B. O'Sullivan, L. Murphy, Increasing task consolidation efficiency by using more accurate resource estimations, Future Generation Comp. Syst. 56 (2016) 407–420. doi:10.1016/j.future.2015.08.018.
URL http://dx.doi.org/10.1016/j.future.2015.08.018

[11] Y. Li, X. Tang, W. Cai, Let's depart together: Efficient play request dispatching in cloud gaming, in: 13th Annual Workshop on Network and Systems Support for Games, NetGames 2014, Nagoya, Japan, December 4-5, 2014, 2014, pp. 1–6. doi:10.1109/NetGames.2014.7008968.
URL http://dx.doi.org/10.1109/NetGames.2014.7008968

[12] F. L. Pires, B. Barán, Multi-objective virtual machine placement with service level agreement: A memetic algorithm approach, in: Proceedings of the 2013 IEEE/ACM 6th International Conference on Utility and Cloud Computing, UCC '13, IEEE Computer Society, Washington, DC, USA, 2013, pp. 203–210. doi:10.1109/UCC.2013.44.
URL http://dx.doi.org/10.1109/UCC.2013.44

[13] G. Wu, M. Tang, Y.-C. Tian, W. Li, Energy-Efficient Virtual Machine Placement in Data Centers by Genetic Algorithm, Springer Berlin Heidelberg, Berlin, Heidelberg, 2012, pp. 315–32. doi:10.1007/978-3-642-34487-9_39.
URL http://dx.doi.org/10.1007/978-3-642-34487-9_39

[14] J. T. Piao, J. Yan, A network-aware virtual machine placement and migration approach in cloud computing, in: Proceedings of the 2010 Ninth International Conference on Grid and Cloud Computing, GCC '10, IEEE Computer Society, Washington, DC, USA, 2010, pp. 87–92. doi:10.1109/GCC.2010.29.
URL http://dx.doi.org/10.1109/GCC.2010.29

[15] Y. Ho, P. Liu, J.-J. Wu, Server consolidation algorithms with bounded migration cost and performance guarantees in cloud computing., in: UCC, IEEE Computer Society, 2011, pp. 154–161.
URL http://dblp.uni-trier.de/db/conf/ucc/ucc2011.html#HoLW11

[16] D. S. Dias, L. H. M. K. Costa, Online traffic-aware virtual machine placement in data center networks.

[17] M. D. Cauwer, D. Mehta, B. O'Sullivan, The temporal bin packing problem: An application to workload management in data centres, in: 28th IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2016, San Jose, CA, USA, November 6-8, 2016, IEEE Computer Society, 2016, pp. 157–164. doi:10.1109/ICTAI.2016.0033.
URL http://dx.doi.org/10.1109/ICTAI.2016.0033

[18] E. Coffman Jr., J. Csirik, G. Galambos, S. Martello, D. Vigo, Bin packing approximation algorithms: Survey and classification, in: P. M. Pardalos, D.-Z. Du, R. L. Graham (Eds.), Handbook of Combinatorial Optimization, Springer New York, 2013, pp. 455–531. doi:10.1007/978-1-4419-7997-1_35.
URL http://dx.doi.org/10.1007/978-1-4419-7997-1_35

[19] P. V. Hentenryck, R. Bent, E. Upfal, Online stochastic optimization under time constraints, Annals OR 177 (1) (2010) 151–183. doi:10.1007/s10479-009-0605-5.
URL http://dx.doi.org/10.1007/s10479-009-0605-5

[20] G. Gutin, T. R. Jensen, A. Yeo, Batched bin packing, Discrete Optimization 2 (1) (2005) 71–82. doi:10.1016/j.disopt.2004.11.001.
URL http://dx.doi.org/10.1016/j.disopt.2004.11.001

[21] Z. Ivkovic, E. Lloyd, Fully dynamic bin packing, in: S. Ravi, S. Shukla (Eds.), Fundamental Problems in Computing, Springer Netherlands, 2009, pp. 407–434. doi:10.1007/978-1-4020-9688-4_15.
URL http://dx.doi.org/10.1007/978-1-4020-9688-4_15

[22] S. Berndt, K. Jansen, K. Klein, Fully dynamic bin packing revisited, CoRR abs/1411.0960.

[23] M. NoroozOliaee, B. Hamdaoui, M. Guizani, M. B. Ghorbel, Online multi-resource scheduling for minimum task completion time in cloud servers, in: Computer Communications Workshops (INFOCOM WKSHPS), 2014 IEEE Conference on, 2014, pp. 375–379. doi:10.1109/INFCOMW.2014.6849261.

[24] J. Csirik, D. S. Johnson, C. Kenyon, P. W. Shor, R. R. Weber, A self organizing bin packing heuristic, in: Algorithm Engineering and Experimentation, International Workshop ALENEX '99, Baltimore, MD, USA, January 15-16, 1999, Selected Papers, 1999, pp. 246–265. doi:10.1007/3-540-48518-X_15.
URL https://doi.org/10.1007/3-540-48518-X_15

[25] C. C. Lee, D. T. Lee, A simple on-line bin-packing algorithm, J. ACM 32 (3) (1985) 562–572. doi:10.1145/3828.3833.
URL http://doi.acm.org/10.1145/3828.3833

[26] E. Danna, E. Rothberg, C. L. Pape, Exploring relaxation induced neighborhoods to improve mip solutions, Mathematical Programming 102 (1) (2005) 71–90. doi:10.1007/s10107-004-0518-7.
URL http://dx.doi.org/10.1007/s10107-004-0518-7

[27] E. Danna, E. Rothberg, C. L. Pape, Exploring relaxation induced neighborhoods to improve mip solutions, Mathematical Programming 102 (1) (2005) 71–90.

Vincent Armant is a Senior Post-Doc researcher at Insight Center for Data Analytics at the University College Cork, in Ireland. He is a Doctor in computer science of Paris Sud University (FRANCE). His main interest is to design complex systems and solving large problems in the area of data analysis and optimization, knowledge discovery, distributed reasoning using optimization techniques.

Milan De Cauwer received his Bachelors in Computer Science in July 2010 and a masters degree in combinatorial optimisation and operations research in 2012 from the Department of Computer Science, Faculty of Sciences and Technologies, University of Nantes, France. He started a PhD program in 2012, University College Cork, Ireland, with a focus on optimisation methods for data centers.

Ken Brown joined UCC Computer Science Department as a senior lecturer in 2003. Prior to that he was a lecturer at the University of Aberdeen, a Research Fellow at Carnegie Mellon University, and a Research Associate at the University of Bristol. His research interests are in the application of AI, optimisation and distributed reasoning, with a particular focus on wireless networks. He is a Co-Principal Investigator/Group Leader in Insight: Centre for Data Analytics

Barry O'Sullivan is Professor of Constraint Programming and Director of the Insight Centre for Data Analytics at University College Cork. He is a Fellow of ECCAI, the European Coordinating Committee for Artificial Intelligence, and a Senior Member of the Association for the Advancement of Artificial Intelligence. Professor O'Sullivan was President of the International Association for Constraint Programming from 2007 to 2012. Professor O'Sullivan's research focuses on optimisation, decision analytics, and constraint programming.

Semi-Online Task Assignment Policies for Workload Consolidation in Cloud Computing Systems

Vincent Armant, Milan De Cauwer, Ken Brown, Barry OSullivan

August 30, 2017

Highlights:

1. We tackle the challenge of consolidating resource in a semi-online workload management system allocating tasks with uncertain duration to physical servers.
2. We propose a formal framework capturing the semi-online consolidation problem.
3. we propose a new dynamic and real-time allocation algorithm based on the incremental merging of bins
4. We develop an adaptation of standard bin packing heuristics with a local search algorithm for the semi-online context considered here.
5. We provide a systematic study of the impact of varying time-period size and varying the degrees of uncertainty on the duration of incoming tasks
6. Our results show that, around periods of high demand, our best policy saves up to 40% of the resources compared to the other polices.